BIFURCATIONS IN THE LEARNING OF RECURRENT NEURAL NETWORKS

Kenji Doya
doya@crayfish.ucsd.edu
Department of Biology
University of California, San Diego
La Jolla, CA 92093-0322, USA

Abstract

Gradient descent algorithms in recurrent neural networks can have problems when the network dynamics experience bifurcations in the course of learning. The possible hazards caused by the bifurcations of the network dynamics and the learning equations are investigated. The roles of teacher forcing, preprogramming of network structures, and the approximate learning algorithms are discussed.

1 Introduction

Supervised learning in recurrent neural networks has been extensively applied to speech recognition, language processing [2, 5, 6], and the modeling of biological neural networks [1, 11, 16, 18]. Although gradient descent algorithms for recurrent networks are considered as a simple extension to the back-propagation learning for feed-forward networks, there is an essential difference between the learning processes in feed-forward and recurrent networks.

The output of a feed-forward network is a continuous function of the weights if each unit has a smooth output function, such as a sigmoid function. In contrast, the output of a recurrent network can change drastically with an infinitesimal change in the network parameter when it passes through a bifurcation point [7, 13]. Thus, the error surface of a recurrent network may be discontinuous in weight space and gradient descent algorithms may not converge.

2 Gradient Algorithms for Recurrent Networks

First, we summarize the learning algorithms for recurrent neural networks. We will deal with the following continuous-time network model.

$$\tau_i \frac{dx_i(t)}{dt} = -x_i(t) + \sum_{j=1}^{n} w_{ij} y_j(t) + \sum_{j=1}^{m} b_{ij} z_j(t), \quad (1)$$

$$y_i(t) = g(x_i(t)),$$

where $x_i(t)$ and $y_i(t)$ $(i = 1, \ldots, n)$ represent the internal state and the output of the units, $g(\ )$ is a sigmoid function, and $z_j(t)$ $(j = 1, \ldots, m)$ represent the external inputs to the network.

The continuous-time model allows simple mathematical derivations and the results thus obtained can easily be transferred to a discrete-time model by setting $\tau_i = 1$ and substituting $x(t+1) - x(t)$ for $dx(t)/dt$. Furthermore, the behavior of biological neural networks can more realistically simulated by continuous-time models than by discrete-time models.

We assume that the output units are indexed by $1, \ldots, o$, the hidden units are indexed by $o+1, \ldots, n$, and the total error of the network is defined by

$$E(t) = \frac{1}{2} \sum_{i=1}^{o} (y_i(t) - d_i(t))^2, \quad (2)$$

where $d_i(t)$ is the desired output, or the "teacher" signal. The goal of learning is to minimize the average error

$$< E >= \frac{1}{T} \int_0^T E(t) dt. \quad (3)$$

The differential relationship between the state of the hidden units and the output units are given by the linearized system of the network dynamics along its actual trajectory. Let the net input to a unit be

$$u_i(t) = \sum_{j=1}^{n} w_{ij} y_j(t) + \sum_{j=1}^{m} b_{ij} z_j(t).$$

If a small perturbation $\epsilon(t) = (\epsilon_1, \ldots, \epsilon_n)^{\mathrm{T}}$ is added to the input $u(t) = (u_1(t), \ldots, u_n(t))^{\mathrm{T}}$, then the deviation of $x(t) = (x_1(t), \ldots, x_n(t))^{\mathrm{T}}$ can be approximated by the solution of the following linear equation system.

$$\tau_i \frac{dp_i(t)}{dt} = -p_i(t) + \sum_{j=1}^{n} w_{ij} g'(x_j(t)) p_j(t) + \epsilon_i(t), \quad (4)$$

$$p_i(0) = 0, \qquad (i = 1, \ldots, n).$$

Thus the differential relationship $\partial x_i(t)/\partial w_{kl}$ is given by the solution $p_i^{kl}(t)$ of (4) with the input

0-7803-0593-0/92 $3.00 1992 IEEE

$$\epsilon_i^{kl} = \frac{\partial u_i(t)}{\partial w_{kl}} = \begin{cases} y_l(t), & i = k, \\ 0, & i \neq k. \end{cases}$$

Then the gradient of the average error is derived as follows.

$$\frac{\partial < E >}{\partial w_{kl}} = \frac{1}{T}\int_0^T \sum_{i=1}^{o} \frac{\partial E(t)}{\partial x_i(t)} \frac{\partial x_i(t)}{\partial w_{kl}} dt$$

$$= \frac{1}{T}\int_0^T \sum_{i=1}^{o} (y_i(t) - d(t)) g'(x_i(t)) p_i^{kl}(t) dt. \qquad (5)$$

This computation method is called real-time recurrent learning (RTRL) [3, 17].

Another method uses the adjoint equation of (4)

$$\frac{dq_i(t)}{dt} = \frac{1}{\tau_i} q_i(t) - \sum_{j=1}^{n} \frac{w_{ji}}{\tau_j} g'(x_i(t)) q_j(t) - \delta_i(t), \quad (6)$$

$$q_i(T) = 0, \qquad (i = 1, \ldots, n).$$

by using the output error as the input

$$\delta_i(t) = \frac{\partial E(t)}{\partial x_i(t)} = \begin{cases} (y_i(t) - d(t)) g'(x_i(t)), & i \leq o, \\ 0, & i > o. \end{cases}$$

Since the solutions of (4) and (6) satisfies the Green's formula [8], the error gradient can be computed as follows.

$$\frac{\partial < E >}{\partial w_{kl}} = \frac{1}{T}\int_0^T \sum_{i=1}^{n} \delta_i(t) p_i^{kl}(t) dt$$

$$= \frac{1}{T}\int_0^T \sum_{i=1}^{n} q_i(t) \epsilon_i^{kl}(t) dt = \frac{1}{T}\int_0^T q_k(t) y_l(t) dt. \quad (7)$$

This method is called back-propagation through time (BPTT) [12, 13, 14].

3 Problems with Bifurcations of Network Dynamics

The solution of (1) is continuous with respect to the initial state $\boldsymbol{x}(0)$ and the parameters $\{w_{ij}\}$. However, the global structure of the state space, which is more important than specific trajectories, has discontinuities at some points in the parameter space, which are called bifurcation points [7].

Let us consider a recurrent network of a single unit

$$\tau \frac{dx(t)}{dt} = -x(t) + wy(t) + b,$$

$$y(t) = 1/(1 + e^{-x(t)}),$$

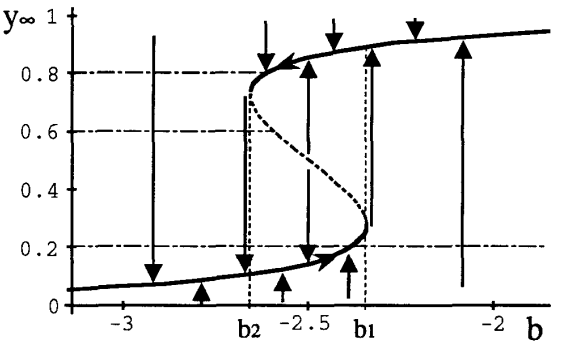

Figure 1: The bifurcation diagram of a self-recurrent network ($w = 5$).

with a fixed recurrent connection $w = 5$. Figure 1 shows the change of the equilibrium y_∞ with the change of the bias b. The solid and dashed curves represent stable and unstable equilibria respectively.

Suppose that the bias is initially $b = 0$, the network state is reset to $y(0) = 0.2$ at the beginning of each training run, and the desired output is $d(t) \equiv 0.8$. At first, b is decreased so that the equilibrium y_∞ approach to 0.8. Then, when b comes to a bifurcation point b_1, as seen in Figure 1, the output becomes abruptly small and the output error increases discontinuously. After that, gradient descent learning oscillates around b_1.

For another example, suppose that "real-time" gradient learning is applied without resetting network state and the desired output is $d(t) \equiv 0.6$. In this case, the bias b cycles between b_1 and b_2. In both cases, the error function does not converge even to a local minimum and the gradient of the error gradient goes to infinity at bifurcation points.

Although these examples use a very simple network and very trivial tasks, similar problems—explosions of the error gradient, jumps of the network state, and non-convergent learning process—can be observed in practical applications of recurrent networks. Actually in many cases of simulations, we found that the error curve did not decrease monotonically even though the learning rate was quite small.

3.1 Instability of learning dynamics

The stability of the linearized system (4) is assured when the trajectory of the network converges to a stable equilibrium. The stability of the adjoint system (6) is the same as that of (4) when it is solved backwards in time from T to 0. However, when an equilibrium bifurcates, the linearized system has an eigenvalue with zero real part. It means that the asymptotic stability of the learning equation is not guaranteed

Furthermore, if an equilibrium changes into a limit cycle through a Hopf bifurcation, the linear system (4) with periodic matrix has a characteristic root $\sigma_i = 1$ [8].

It again implies that the learning system may not be stable. Actually, we observed in computer simulations that the solution of (4) grew exponentially, thus causing destructive change in the weights.

3.2 Non-convergence of learning

As illustrated in the above example, in some sorts of bifurcations, such as saddle-node bifurcations and subcritical Hopf bifurcations, the trajectory of the system can change discontinuously with a small change in the parameter. This can increase the error function in the course of learning.

In some cases, the change of the network parameters can fall into a limit cycle. Furthermore, if a constant learning rate is used in gradient descent, a very steep error gradient near bifurcation points causes a very long jump in the parameter space. It is possible that a network undergoes almost random jumps in the parameter space until it falls in a favorite basin where gradient descent learning works.

3.3 Confusion of input-output patterns

For many tasks in which the network is supposed to operate as an automaton, its state space must have multiple attractor basins to store the distinct internal states. However, if the connection weights are set to small random values at the beginning of learning, the network has only one attractor basin. This implies that the network must undergo several bifurcations, such as saddle-node and pitchfork bifurcations, on which the gradient descent learning may get into trouble. Until the network has enough numbers of attractor basins, the network often fails to discriminate input sequences and learns only the average value of the output.

External inputs to the network can change the structure of the state space and the confusion of different input-output pairs may be avoided. But it is not clear under what conditions a network can achieve the correct global structure of the state space by only being shown sample input-output trajectories.

4 Some Means to Overcome Bifurcation Problems

A negative choice to avoid the hazards of bifurcation is to restrict the network dynamics to have a single global point attractor. The convergence of gradient descent learning has been proved under this condition [10]. A recurrent network with a single or a few attractor basins can be used as a non-linear adaptive filter [1, 11] or as the source of short term memory [18]. However, in order to exploit the full potential of recurrent networks, we must train the network beyond the bifurcation boundaries.

4.1 Non-recurrent learning algorithms

One simple way to avoid the instability of the learning dynamics is to use a non-recurrent learning rules. Feedforward approximations of recurrent dynamics were successfully used in sequence generation [3, 9] and sequence prediction tasks [2, 5]. Those learning rules are derived from (4) by setting the weights w_{ij} to zero except for those of the hidden-to-output connections $(i = 1, \ldots, o; j = o+1, \ldots, n)$. They need only $O(on^2)$ computations, whereas RTRL requires $O(n^4)$ computations.

4.2 Teacher Forcing

It has been reported that a technique called "teacher forcing" is required in some learning tasks. It replaces the actual output of the network with the desired output;

$$\tau_i \frac{dx_i(t)}{dt} = -x_i(t) + \sum_{j=1}^{o} w_{ij} d_j(t) + \sum_{j=o+1}^{n} w_{ij} d_j(t) + \sum_{j=1}^{m} v_{ij} z_j(t) \qquad (8)$$

$$y_i(t) = g(x_i(t))$$

If the actual outputs $y_i(t)$ $(i = 1, \ldots, o)$ are made equal to $d_i(t)$ by learning, the solution of (8) is same to that of the non-forced system (1). Although the stability of the non-forced solution is not theoretically guaranteed, they were found to be stable in most of the computer simulations [3, 13, 17].

It has been shown that teacher forcing is essential in the learning of oscillatory patterns [3, 17]. One of the roles of teacher forcing is the synchronization of the network dynamics to the teacher signal. Suppose the network has nearly learned the desired waveform and is running autonomously. Unless the frequency of the network oscillation is exactly equal to that of the teacher signal and the initial phase is exactly matched, the phases of the network output and the teacher signal are not kept equal. This makes an apparent large error and destroys the nearly desired structure of the network.

In this sense, not all the output units have to be teacher-forced. Moreover, a "weak" teacher forcing is possible by replacing $y_i(t)$ with

$$(1 - \alpha) y_i(t) + \alpha d_i(t) \qquad (0 \leq \alpha \leq 1). \qquad (9)$$

A stable solution of a weakly forced system is more likely to remain stable in the unforced system than those of fully forced systems.

Another role of teacher forcing is to avoid the problems with bifurcations by decreasing the degree of freedom of the network dynamics. In the training of associative memory using gradient descent learning, teacher

forcing was required to avoid degeneration of the network trajectories with different training patterns [13]. In the learning of oscillatory patterns, teacher forcing avoids the problems with Hopf bifurcations as mentioned above.

Teacher forcing can also be considered as a method to estimate the desired network state. The additional input made by the generalized teacher forcing (9) can be expressed as

$$v_i(t) = -\alpha(y_i(t) - d_i(t)) = -\alpha\frac{\partial E(t)}{\partial y_i(t)}. \qquad (10)$$

This can be regarded as a gradient descent algorithm for the network state and can also be applied to hidden units using the same gradient computation for the weights.

4.3 Preprogramming of the state space

If the required structure of the state space is known a priori, a practical approach is to preprogram the structure and the connection weights of the networks.

For example, the learning of multiple limit cycle attractors, each with specific waveforms, was successfully carried out by preprogramming the network to have multiple attractor basins[4]. It has been demonstrated that chaotic oscillations can be learned in a structured network of local recurrent circuits with different time constants [15].

5 Conclusions

In multi-layer feedforward networks, the existence of local minima in the weight space has been supposed to be practically no problem [14]. However, in recurrent networks, the basis of gradient descent learning, the continuity and the differentiability of the error surface, can be violated. Therefore, more consideration should be given to the view point of dynamical systems and bifurcation theory.

In spite of the possible problems, there are many examples where gradient learning successfully trained a recurrent network to model complex dynamical behaviors [6, 15, 17]. It is important to investigate the underlying conditions for these successful cases and the reasons for (mostly unpublished) unsuccessful results.

Acknowledgements

The author is grateful to P.F. Rowat and A.I. Selverston for their discussions and help. This research was partially supported by ONR grant N00014-91-J-1720.

References

[1] Anastasio, T.J. 1991. Neural network models of velocity storage in the horizontal vestibulo-ocular reflex. *Biological Cybernetics*, **64**, 187–196

[2] Cleeremans, A., Servan-Schreiber, D., and McClelland, J.L. 1989. Finite state automata and simple recurrent networks. *Neural Computation*, **1**, 372–381.

[3] Doya, K. and Yoshizawa, S. 1989. Adaptive neural oscillator using continuous-time back-propagation learning. *Neural Networks*, **2**, 375–385.

[4] Doya, K. and Yoshizawa, S. 1989. Memorizing oscillatory patterns in the analog neuron network. *Proceedings of International Joint Conference on Neural Networks 1989 in Washington DC*, **I**, 27–32.

[5] Elman, J.L. 1990. Finding structure in time. *Cognitive science*, **14**, 179–211.

[6] Giles, C.L., Miller, C.B., Cheng, D., Chen, H.H., Sun, G.Z., and Lee, Y.C. 1992. Learning and extracting finite state automata with second-order recurrent neural networks. *Neural Computation*, **4**.

[7] Guckenheimer, J. and Holmes, P. 1983. *Nonlinear oscillation, dynamical systems, and bifurcations of vector fields.* Springer Verlag.

[8] Hartman, P. 1982. *Ordinary Differential Equations*, Birkhäuser.

[9] Jordan, M.I. 1989. Serial order: A parallel distributed processing approach. In: J. L. Elman and D. E. Rumelhart (eds.) *Advances in Connectionist Theory: Speech*, Erlbaum.

[10] Kuan, C., Hornik, K., and White, H. 1990. Some convergence results for learning in recurrent neural networks. *UCSD Department of Economics Discussion Paper*, 90-42.

[11] Lockery, S.R., Fang, Y., and Sejnowski, T.J. 1990. A dynamic neural network model of sensorimotor transformation in the leech. *Neural Computation*, **2**, 274–282.

[12] Pearlmutter, B.A. 1989. Learning state space trajectories in recurrent neural networks. *Neural Computation*, **1**, 263–269.

[13] Pineda, F.J. 1988. Dynamics and architecture for neural computation. *Journal of Complexity*, **4**, 216–245.

[14] Rumelhart, D.E., Hinton, G.E., and Williams., R.J. 1986. Learning internal representations by backpropagating errors. *Nature*, **323**, 533–536.

[15] Sato, M., Joe, M., and Hirahara, T. 1990. APOLONN brings us to the real world: Learning nonlinear dynamics and fluctuations in nature, *Proceedings of International Joint Conference on Neural Networks 1990 in San Diego*, **I**, 581-587.

[16] Tsung, F., Cottrell, G.W., and Selverston, A.I. 1990. Some experiments on learning stable network oscillations. *Proceedings of International Joint Conference on Neural Networks 1990 in San Diego*, **I**, 169–174.

[17] Williams, R.J. and Zipser, D. 1989. Experimental analysis of the real-time recurrent learning algorithm, *Connection Science*, **1**, 87–111.

[18] Zipser, D. 1991. Recurrent network model of the neural mechanism of short-term active memory. *Neural Computation*, **3**, 179–193.